CS login: _____ Seat Number: _____

CSc 372 Mid-Term Examination
October 17, 2006

READ THIS FIRST

Read this page now but do not turn this page until you are directed to do so. Go ahead and fill in your login and seat number.

This is a 60-minute exam with a total of 100 points of regular questions and an extra credit section.

You are allowed no reference materials whatsoever.

If you run out of room, write on the back of a page. DO NOT use sheets of your own paper.

If you have a question, raise your hand. One of us will come to you. DO NOT leave your seat!

There is an exam-wide restriction: You may not use regular expressions or hashes in a Ruby solution.

If you have a question that can be safely resolved with a minor assumption, state the assumption and proceed. Examples:

> Assuming `String.sub` is `string * int -> char`
> Assuming `tl []` returns `[]`.
> Assuming `String#downcase!` imperatively converts all capitals to lower case.

BE CAREFUL with assumptions that dramatically change the difficulty of a problem. If in doubt, ask a question.

Unless explicitly prohibited on a problem you may use helper functions/methods.

Don't waste time by creating solutions that are more general, more efficient, etc. than required. Take full advantage of whatever assumptions are stated.

As a broad rule when grading, we consider whether it would be likely if the error would be easily found and fixed if one were able to run it. For example, something like `i + x` instead of `i + x.to_i`, or forgetting a `chomp` will be typically a minor deduction at worst. On the other hand, an error that possibly shows a fundamental misunderstanding, such as a `yield` with no argument for a block that expects one, will often lead to a large deduction.

Feel free to use abbreviated notation such as I often use when writing on the Elmo. For example, you might use a ditto instead of writing out the function name for each case or abbreviate a function/method name to `a_b_c` or `ABC`. Don't worry about matching parentheses at the end of a line—just write plenty and we'll know what you mean.

You need not include any explanation in an answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that will certainly earn no credit.

When told to begin, double-check that your name is at the top of this page, and then **put your initials, or some other distinctive mark, in the lower right hand corner of each page**.

BE SURE to check that you have all 12 pages.

When you have completed the exam, enter your name on the exam sign-out log and then hand your exam to the instructor.

Problem 1: (5 points)

The instructor often says "In ML we never change anything; we only make new things." What does he mean by that?

Problem 2: (5 points)

(a) Write an ML function `fun firstLast(L)` that returns a tuple consisting of the the first and last elements of the list `L`. Assume `L` has at least one element. **Restriction: You may use helper functions but you may use no functions other than functions you write.** Hint: Write a helper function `last(L)`.

(b) What is the type of `firstLast`?

Problem 3: (5 points)

Consider a list of one-element lists:

```
[[10], [2], [5], [77]]
```

Imagine an ML function `f` that will "flatten" such lists, like this:

```
- f [[10], [2], [5], [77]];
 val it = [10,2,5,77] : int list
```

Using at most eight (8) characters, fill in the blank below to create `f`:

```
val f = _ _ _ _ _ _ _ _
```

Problem 4: (6 points)

Write an ML function `eo(L)` that returns a list consisting of every other element of the list `L`. (That is, the 2nd, 4th, 6th, 8th, ... elements.) **Restriction: You may use helper functions but you may use no functions other than functions you write.** Examples:

```
- eo [1,7,9,12];
val it = [7,12] : int list

- eo [5,3,10];
val it = [3] : int list

- eo (iota 10);
val it = [2,4,6,8,10] : int list
```

Problem 5: (12 points)

Without writing a function that is directly or indirectly recursive, write an ML function `ints_to_string(L)` that produces a string representation of `L`, an `int list`. As shown below, the values are separated by a comma and a space. If you wish, you may use `Int.toString`, of type `int -> string`, to convert individual values. (It produces `"~3"` for `~3`.) Examples:

```
- ints_to_string([10,5,3,~3]);
val it = "10, 5, 3, ~3" : string

- ints_to_string([123]);
val it = "123" : string

- ints_to_string([]);
val it = "" : string
```

Problem 6: (15 points)

Without writing a function that is directly or indirectly recursive, write an ML function `show_lists(L)` of type `(string * int list) list -> unit` that prints the contents of `L`, prefixing each `int list` with the specified label. Example:

```
- show_lists [("a",[3,5,1]),("list2",[]),("c",(iota 10))];
a: 3, 5, 1
list2: <empty>
c: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
val it = () : unit

- show_lists [("Result(s)", [10])];
Result(s): 10
val it = () : unit

- show_lists [];
val it = () : unit
```

Note that if a tuple's `int list` is empty, the string `"<empty>"` is printed. Be sure that your solution does not produce any trailing whitespace—the last digit, or `"<empty>"`, should be immediately followed by a newline.

If you wish, you may make use of `ints_to_string`, from the previous problem. (Assume a working version.)

Problem 7: (10 points)

Consider a folding that operates on an `int list` with an even number of values, all greater than zero, and produces a list of pair-wise sums: [*1st+2nd, 3rd+4th, ..., (N-1)th+Nth*]

Example:

```
- foldr f [] [5,7, 3,4, 9,8];
val it = [12,7,17] : int list

- foldr f [] [10,20];
val it = [30] : int list

- foldr f [] [4,1, 3,2, 2,3, 4,1];
val it = [5,5,5,5] : int list
```

In this problem you are to write a function `f` such that the above foldings work as shown.

Hint: Remember the technique of writing out a fixed expansion of the calls, like `f(e1,f(e2,f(e3,[])))`, to help develop the function to fold with.

Keep in mind that you are writing a function to use with `foldr`, not a function that performs this pair-wise summing directly.

Hint: I think that to create an exam-size solution you must take advantage of the fact that all values are greater than zero. If you don't quickly see an approach, you may be wise to skip this problem and come back to it if time permits.

Problem 8: (4 points)

(a) Is the following ML function declaration valid? If valid, what is the type of `f1`? If not valid, explain why it is not valid.

```
fun f1 () f2 () = [f2];
```

(b) Write an ML function `g` whose type is

```
int -> (int list * int) -> (string list) -> (bool * real)
```

UNLIKE `ftypes.sml` on the first assignment, there are no restrictions on this problem.

Problem 9: (4 points)

(a) Using nothing but a `val` binding and the composition operator, create an ML function `f(s)` that returns a copy of the string `s` with the first and last characters removed. Assume that `size(s) >= 2`.

Examples:

```
- f "string";
val it = "trin" : string

- f "ab";
val it = "" : string

- f "abc";
val it = "b" : string
```

The requirements imply that your solution must look like this:

```
val f = f1 o f2 o ... o fN.
```

(b) As you've defined it, what is the type of `f`? (Don't forget to properly account for the intermediate functions in the composition.)

Problem 10: (16 points)

When showing examples of interaction with `irb` it saves space to put both the expression and the result on the same line. However, it is tedious to left-align the results in a column.

Write a Ruby program that reads from standard input a script of interaction with `irb` and combines the expressions and results on a single line, vertically aligning the results, based on the longest input expression. Each combined line is followed by a blank-line.

An optional command line argument specifies the number of spaces between the end of the longest input expression and its result. (Assume the specified spacing is good, not `"x"`, `"5x"` or `"-3"`, for example.) If no argument is specified, one space is used. Example:

```
% cat irbfmt.1
>> 3+4
=> 7
>> a = %w{words in array}
=> ["words", "in", "array"]
>> a.max
=> "words"
% ruby irbfmt.rb 3 < irbfmt.1   # NOTE: 3 spaces before "=>"
>> 3+4                      => 7

>> a = %w{words in array}   => ["words", "in", "array"]

>> a.max                    => "words"

%
```

Because expressions and results should always be paired, it is an error if the number of input lines is odd:

```
% cat irbfmt.2
>> 3*7
=> 21
>> it.class
% ruby irbfmt.rb < irbfmt.2
Error: short input
```

Write your answer below or use the whole page that follows.

Don't forget to handle the optional command-line argument.

(Space for `irbfmt.rb`)

Problem 11: (2 points)

Write a Ruby program that reads all lines from standard input and prints them on standard output in reverse order, last line first, first line last. Assume there is at least one line and that the file ends with a newline.

```
% cat revlines.1
reverse
the
order
% ruby revlines.rb < revlines.1
order
the
reverse
%
```

For two points of extra credit, have less than 25 characters in your solution. (No abbreviations on this one!) To help you pursue this option, here are 24 blanks:

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Problem 12: (3 points)

Here is a line of code from a Ruby method:

```
line = (gets || return)
```

Imagining the reader to be a Java programmer with no knowledge whatsoever of Ruby, explain its operation in each of the possible cases that might arise when it executes. Ignore the open-command-line-arguments-as-files-and-read-them behavior of `gets`.

Problem 13: (8 points)

Write a Ruby iterator named `upto_limit(a, limit)`. The argument `a` is an array of integers; `limit` is an integer. `upto_limit` yields the values of `a` in turn (`a[0]`, `a[1]`, ...) continuing while the sum of the yielded values is less than or equal to `limit`. `upto_limit` returns `a`.

```
>> upto_limit([1,2,3], 5) { |x| puts x }
1
2
=> [1, 2, 3]

>> upto_limit([1,1,1,1],3) { |x| puts x }
1
1
1
=> [1, 1, 1, 1]

>> sum = 0
=> 0
>> upto_limit([10,20,30,40], 1000) { |x| sum += x }
=> [10, 20, 30, 40]
>> sum
=> 100

>> upto_limit([0,0,0,1], 0) { |x| puts x }
0
0
0
=> [0, 0, 0, 1]
```

Problem 14: (5 points)

Write a Ruby method `extract(s, m, n)` that extracts a portion of a string that represents a hierarchical data structure. m is a major index and n is a minor index. Major sections of the string are delimited by slashes and are composed of minor sections separated by colons. Here is a sample string:

```
/a:b/apple:orange/10:2:4/xyz/
```

It has four major sections which in turn have two, two, three and one minor sections. A call such as `extract(s, 3, 2)` should locate the third major section (`"10:2:4"` in the string above) and return the second minor section therein (`"2"`). If either section number is out of bounds, `extract` returns `nil`. Assume that `m` and `n` are greater than zero and that `s` is well-formed.

```
>> s = "/a:b/apple:orange/10:2:4/xyz/"
=> "/a:b/apple:orange/10:2:4/xyz/"

>> extract(s,1,1)    => "a"

>> extract(s,1,2)    => "b"

>> extract(s,3,3)    => "4"

>> extract(s,4,1)    => "xyz"

>> extract(s,4,2)    => nil

>> extract(s,10,1)   => nil
```

Hint: `Assume that "|20|1|300|".split("|")` produces `["20", "1", "300"]`.

Extra Credit Section (one point each unless otherwise indicated)

(1) Imagine that you have an ML function named `f` and a Ruby method named `f`. Does `[f]` produce an analogous result in both languages? If not, how do the results differ?

(2) For up to three points name three programming languages developed at the University of Arizona and give an example of a valid expression in each that involves an operator.

(3) For one point each, write `curry` and `uncurry` in ML.

(4) Assuming that `s` is a string, what is a Ruby expression that produces the same effect as `s.dup` but is both shorter and more difficult to type?

(5) What element of Ruby most closely corresponds to an anonymous function in ML?

(6) Simplify this Ruby expression: `if a[0] == nil then false else true end`

(7) Cite a contribution to knowledge made by Ralph Griswold.

(8) (Up to five points.) Offer some intelligent observations about the applicability of type deduction, or something similar, in Ruby.